JPEG Encoder IP Core Setup Guide

for Xilinx Vivado

VISENGI

Hardware & Software Engineering

www.visengi.com

Revision History

Rev.	Date	Description
1.0	2017-11-24	Initial documentation

Table of Contents

Setting Up

The following guide describes how to set up the Xilinx Vivado drag'n'drop instance deliverables for **implementing VISENGI's JPEG Encoder IP core on Xilinx Vivado projects**.

Requirements

- Xilinx Vivado (WebPack or Full Editions)

List of deliverables

Three versions of the same VISENGI JPEG Encoder (JPEGE) IP core are delivered, the only difference between versions being the bus used on the Data I/O Interface (for Pixel Input and Coded Output):

- **Folder "jpeg[x]_encoder_v1_00_a":** JPEG Encoder Vivado Block Design instance files, with AXI4-Lite for the configuration interface and AXI3* (memory-mapped) interfaces for Pixel Input and Coded Output.

- **Folder "jpeg[x]_encoder_st_v1_00_a":** JPEG Encoder Vivado Block Design instance files, with AXI4-Lite for the configuration interface and AXI4-Stream interfaces for Pixel Input and Coded Output. The pixel input is compatible with AXI4-ST Video IP interfaces as defined by Xilinx in user guide UG934.

- **Folder "jpeg[x]_encoder_stin_v1_00_a":** JPEG Encoder Vivado Block Design instance files, with AXI4-Lite for the configuration interface, AXI3* for Coded Output, and AXI4-Stream interface only for Pixel Input. The pixel input is compatible with AXI4-ST Video IP interfaces as defined by Xilinx in user guide UG934.

*AXI3 buses are forward compatible towards AXI4 interconnects.

Note: the optional "x" in the component names above indicates that the JPEG Encoder may be capable Extended Baseline mode too (i.e. > 24bpp), if the IP core licensed is "JPEG Extended".

Evaluation limits

If these files were downloaded from VISENGI's website, their purpose is to show how simple it is to embed a hardware JPEG Encoder into any design.

As such, please note that the delivered folders may be missing the netlist file itself. Hence, the design can be fully connected/implemented but not simulated, nor synthesized into an FPGA bitstream.

The licensed version of the IP core consists of the same files, but with an added “netlist” folder, allowing bitstream generation without changes to the Block Design system implemented here.

Setting up the deliverables

In order to design systems including VISENGI's JPEG Encoder IP core on Xilinx Vivado, the simplest way is to use Vivado Block Design GUI tool, the accompanying folders must be extracted to a user folder and added to a project's IP repository by following these steps:

1) Open Xilinx Vivado SW and either click "Create New Project" or "Open Project" if one already exists for the target Xilinx board.

2) Once the project is created/open, go to "Project Settings" in the "Tools" menu .

3) On the "Project Settings" window click on the "IP" icon and then on "Add Repository..."

4) Select the folder that contains VISENGI's Xilinx IP description folders (jpegx_encoder_*) and click Ok .

5) The IP components in the repository ("VISENGI JPEG Encoder ...") should appear at the bottom, click Ok .

Please note that the folder where the jpegx_encoder_* subfolders were extracted must not be deleted or the IP Cores will not be accessible (Vivado does not copy them into its IP library).

Instantiation

In order to instantiate the JPEGE IP core into a Xilinx Block Design, the following steps should be followed:

1. Open Xilinx Vivado SW, and then the project where the JPEGE is to be integrated.
2. Select "Open Block Design" (or "Create Block Design" if none exists)
3. Once the block design is open, right click on the background and click on "Add IP..."
4. In the search box that appears, type "VISENGI" and the JPEG Encoders will appear
5. Double click the desired JPEG Encoder (with AXI or with AXI4-Stream interfaces) or drag-and-drop it onto the block design's background
6. The new instance will now appear in the design without any connections.

AXI vs AXI4-Stream Interfaces

The three JPEGE instances delivered feature the most common interfacing possibilities of the IP core's Data I/O interface.

Firstly, let's review the common interfaces:

- **Clock and reset:** the single clock (clk) and reset (rst) input ports are shared by all interfaces and internal logic. The reset is active-low Synchronous.

- **Configuration Interface (S_AXI):** AXI4-Lite slave with a 32 bits data interface and 12 bits addressing to control all the necessary parameters of encoding through a small Configuration register set. This is a low speed interface.

- **Interrupt Interface (JPEGEX_int_o):** A rising-edge interrupt is available, signaling when image compression has finished. Idle/busy status can also be checked through the Configuration register set.

The AXI3/4 interfaces contain DMA logic to enable connecting them directly to high speed buses towards external memory (or directly to a memory controller port) for data I/O.

On the other hand, AXI-4 Stream interfaces are meant usually to be connected as part of a pixel processing pipeline, for example.

The three different combinations of I/O buses available are:

1) **jpeg_encoder / jpegx_encoder** (JPEG Encoder AXI I/O):

The "M_AXI" bus (used to read **input pixels** and write the coded **JPG output**) is an **AXI3** Master, with a 32 bits data width (**little-endian**), embedded DMA engine for direct connection to a memory controller, and user-set addressing parameters through the Configuration register set. The "jpeg_encoder" is a Baseline JPEG Encoder (i.e. 24 bpp), whereas the "jpegx_encoder" is a Baseline and Extended Baseline (up to 36 bpp) encoder.

2) **jpeg_encoder_st / jpegx_encoder_st** (JPEG Encoder AXI4-ST I/O)**:**

The **Pixel Input AXI4-Stream** Slave Interface is named "S_AXIS_PIX_IN" and can be connected to a Video IP AXI4-Stream Interface that feeds pixels in a row-wise manner (such as from an image sensor or video input).

Data width is <u>24 bits</u> (for jpeg_encoder_st, <u>JPEG Encoder Baseline</u>) for 24bpp pixels. Or else data width is <u>40 bits</u> (for jpegx_encoder_st, <u>JPEG Encoder Extended</u>), where pixels are LSB aligned as per AXI4-ST Video IP definition from Xilinx User Guide UG934. That is, bits [35:24] are for R/Cr component, bits [23:12] for B/Cb component, and bits [11:0] for G/Y component, as per Xilinx AXI4-Stream Video IP definition UG934.

If input is <12 bits per sample, the components are MSb aligned (example for 24bpp: R is at [35:28], G at [23:16], and B at [11:4]) and the LSbs of each color are to be 0-padded.

The Coded **JPG Output AXI4-Stream** Master Interface is named "M_AXIS_JPG_OUT". It outputs 32 bits words in a sequential manner (bytes in little endian order, as per AXI spec.). If they are written incrementally into a single file, the result is a .jpg file. Data width is 32 bits **little-endian** (i.e. 1st byte in .jpg file is the LSB, 4th byte is at the MSB). The last word of the .jpg file is padded if necessary (at its MSBs).

3) **jpeg_encoder_stin** / **jpegx_encoder_stin** (JPEG Encoder AXI4-ST In and AXI Out):

This third version features a mix of the above:

The **Pixel Input AXI4-Stream** Slave Interface is named "S_AXIS_PIX_IN" and is exactly the same as for jpeg_encoder_st (24 bits) / jpegx_encoder_st (40 bits) above.

The "M_AXI" bus (write only for the Coded **JPG Output**) is an **AXI3** Master, with a 32 bits data width (**little-endian**), embedded DMA engine for direct connection to a memory controller, and user-set addressing parameters through the Configuration register set.

Interfacing

The following interfacing scheme is recommended for maximum performance, since bottlenecks in the Data AXI interfaces are about the only way to slow down the JPEG Encoder.

In order to allow the IP core to maintain its constant pixel throughput, the data I/O AXI interface must be connected to a high speed slave with direct access to memory.

CPU/DDR Configuration:

In this **connection example** we will suppose a **Zynq 7xxx** FPGA, that is: an FPGA with embedded ARM CPUs.

> NOTE: the JPEGE IP core is fully autonomous, the use of a CPU in this example is just for convenience. It is only employed to configure the compression parameters (frame width/height, quality, DMA addresses, etc) through the AXI4-Lite interface. However, a soft-core CPU (such as a Microblaze/Nios/...) or even a simple FSM, can be used for the same purpose (on non-ARM enabled FPGAs).

As stated, the CPU is supposed to be the "ZYNQ7 Processing System" (ARM) which should already appear as a block in the Block Design in the "Diagram" tab.

Double click on it to open the "Re-customize IP" window and:

- In the **"PS-PL Configuration"** section, open the **"GP Master AXI interface"** drop-down and make sure that the "M AXI GP0 interface" is marked. This is where the AXI4-Lite Configuration interface (S_AXI) will go.

- In the same section, open the **"HP Slave AXI interface"** drop-down and make sure that "S AXI HP0 interface" is marked (Important: please make sure that if your design uses another interface then "0 and 2" or "1 and 3" are enabled, for performance reasons). Open it and make sure that "DATA WIDTH" is 32.

- In the **"Interrupts"** section, make sure that there is a mark on the checkbox labeled "Fabric Interrupts", open the drop-down, and then open the item labeled **"PL-PS Interrupt Ports"**, then make sure that the checkbox labeled "IRQ_F2P[15:0]" is marked. This is where the IP core Interrupt output (JPEGEX_int_o) will connect to the CPU.

Click OK to close this window and for the changes to take effect.

> NOTE: On a non-ARM enabled FPGA, the S_AXI AXI4-Lite low speed bus should be connected to a soft-core CPU, or a suitable FSM, for configuration; whereas the AXI3 I/O Data interface should be connected to a DDR Memory Controller.

Interconnection:

Now all the necessary endpoints should be available in the Vivado Block Design's "Diagram" tab. To properly connect the IP core these instructions should be followed.

First of all Add the IP corresponding to the JPEG Encoder IP core. In this case we will be using the one with default AXI interfaces (i.e. jpeg_encoder or jpegx_encoder).

To create the connections between ports/buses: click on the source port (short line coming out of the block with the indicated port name) and drag it to the target port, a routed line will appear connecting both.

- The **"rst"** input port **MUST BE** connected to an active-low **SYNCHRONOUS reset source**. This is very important, since connecting it to an asynchronous reset (i.e. aresetn*) will yield significantly worse timings.

  On a Zynq device this should be the "FCLK_RESET0_N" output of the Xilinx "ZYNQ7 Processing System" block.

- The **"clk"** input port must be connected to a rising-edge active clock source.

  On a Zynq device this could be, for example, port FCLK0.

- It is recommended to use the **"JPEGEX_int_o"** rising-edge active interrupt signal.

  On a Zynq device, connect it to the IRQ_F2P port.

- The **"S_AXI"** bus (used to access the IP core's Configuration Registers) is a standard AXI4-Lite interface of data width 32 bits.

On a Zynq device it can be directly connected to the PS' M_AXI_GPn port. If there are multiple AXI4-Lite devices connected to that bus, or said AXI_GP Master's clock is different than FCLK0, then a Xilinx "AXI Interconnect" block will be necessary to connect both.

- The **“m_axi”** bus (used to read input pixels and write the compressed output file) is a standard AXI3 interface of data width 32 bits, which can be directly connected to a memory controller, since it already embeds the necessary DMA engine.

On a Zynq device, for maximum performance, it is recommended to connect it to the S_AXI_HP0 port through an AXI Interconnect (never use S_AXI_HP0 and _HP1, or _HP2 and _HP3 since each pair shares the same internal bus).

NOTE: “S_AXI_HP*” buses all connect the FPGA to the CPU's shared DDR memory controller. That sharing between FPGA logic and ARM CPUs poses no problems.

Address Editor:

Finally, click on the “Address Editor” tab and open the drop-down tree until finding an **“jpeg[x]_encoder*”** item, then right click and select **“Auto Assign Address”**.

The automatically assigned addresses correspond to these IP core items:

- Within the **“processing_system7_*”** leaf, find the item **“jpeg[x]_encoder_* S_AXI”** : this is the assigned memory range to access the Configuration register set of the IP core.

- Within the **“jpeg[x]_encoder_*”** leaf, find the item **“processing_system7_* S_AXI_HP0”**: here the total memory range should have been set, usually from “0x0000_0000” to “0x3FFF_FFFF” for 1GB systems. The actual address range used by the IP core is set later in the Configuration register set.

In order to check that the block design is correctly connected, go to the Tools menu and click on "Validate Design" or hit F6.

The system is ready!

For any question on how to best interface this IP core for your application, please do not hesitate to send an email to info@visengi.com